Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Borut Milič

Sistem za nadzor dostopa

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Načrtujte sistem za nadzor dostopa na osnovi RFID značk. Sistem za nadzor dostopa načrtujte na sistemu na čipu STM32F407. Za preverjanje uporabnika uporabite RFID značke in tipkovnico za vnos kode. Vgradno programsko opremo načrtujte v operacijskem sistemu FreeRTOS. Za upravljanje z uporabniki načrtujte podatkovno bazo in aplikacijo na osebnem računalniku.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Borut Milič, z vpisno številko **63050290**, sem avtor diplomskega dela z naslovom:

Vgrajen sistem za kontrolo dostopa na operacijskem sistemu FreeRTOS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. oktobra 2014 Podpis avtorja:

Zahvaljujem se mentorju za uporabne nasvete, ki so mi olajšali izdelavo diplomske naloge.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
RTOS	real-time operating system	operacijski sistem v realnem času
ARM	RISC machine	ARM procesorska arhitektura
RISC	reduced instruction set computer	reduciran nabor ukazov procesorja
RFID	radio frequency identification	radiofrekvenčna identifikacija
TFT	thin-film transistor	tankoplastni tranzistor
PIN	personal identification number	osebna identifikacijska številka
RS232	standard for serial communication	standard serijske komunikacije
TCP	transmission control protocol	protokol za nadzor prenosa
RAM	random access memory	pomnilnik z naključnim dostopom
ROM	read only memory	bralni pomnilnik
LED	light-emmiting diode	svetleča dioda
USB	universal serial bus	univerzalno serijsko vodilo
DSP	digital signal processing	digitalna obdelava signalov
MIPS	million instructions per second	milijon ukazov na sekundo
FPU	floating point unit	enota za operande v plavajoči vejici
DMA	direct memory access	direktni način dostopa do pomnilnika
SPI	serial peripheral interface	serijski periferni vmesnik
CAN	controller area network	področno vodilo CAN
I^2C	inter-integrated circuit	vodilo I^2C
CRC	cyclic redundancy check	ciklično redundantno preverjanje
ADC	analog-to-digital converter	analogno-digitalni pretvornik
SAR	successive approximation	zaporedna aproksimacija
RGB	red-green-blue color model	RGB barvni model

Povzetek

V diplomskem delu je predstavljen sistem kontrole dostopa, sestavljen iz mikrokrmilnika STM32F4, ki temelji na procesorju ARM Cortex-M4, ekrana na dotik, čitalca pasivnih kartic in pretvornega modula RS232/TCP. Poleg omenjene strojne opreme sta del sistema še dva programa, ki se ne izvajata na mikrokrmilniku. Prvi je strežniški program, ki v bazi uporabnikov preverja vrednosti značk kartic in pripadajočih PIN-ov, drugi pa je namenjen pregledu ter beleženju dostopov, lahko pa tudi dodajamo ali brišemo uporabnike. Na mikrokrmilniku teče FreeRTOS, prosta različica operacijskega sistema v realnem času. Kodiranje krmilnika je izvedeno v programskem jeziku C, drugo pa v programskem jeziku Java in C#.

Ključne besede: FreeRTOS, vgrajeni sistemi, kontrola dostopa, RFID, ARM.

Abstract

This diploma paper introduces development and implementation of access control system. It is made of STM32F4 microcontroller based on ARM Cortex-M4 processor, TFT touch screen, RFID card reader and RS232/TCP converter module. Part of the system are also two programms which are running outside of the controller. First, the server side program is used to check RFID tags and belonging PINs, the second one does the logging. Beside logging, its functionality is also adding or removing users from the database. Operating system we are using on microcontroller is FreeRTOS, one of the free versions of real time operating systems. We used programming language C for coding the microcontroller and programming language Java in combination with C# for the rest.

Keywords: FreeRTOS, embedded system, access control, RFID, ARM.

Poglavje 1

Uvod

Z razvojem tehnologije in s konstantnim padanjem cen elektronskih komponent se tudi domačim uporabnikom z nekaj znanja na področju računalništva odpira vse več možnosti za implementacijo sistemov, ki jih v preteklosti doma ne bi ravno pričakovali. Tak sistem je predstavljen v diplomskem delu, in sicer kontrola dostopa.

Bistveni del je zmogljiv STM[1] mikrokrmilnik s cenovno zelo ugodnim procesorjem Cortex-M4. Mikrokrmilniki dandanes niso nič novega, saj so na trgu že dobrih 40 let, se je pa njihova zmogljivost v zadnjih letih bistveno povečala, cena pa vztrajno pada.

Prva komponenta, ki smo jo povezali na mikrokrmilnik, je čitalec pasivnih kartic. Ko uporabnik pride do vrat, svojo identifikacijsko kartico približa čitalcu. Ta prebere unikatno dvanajstmestno kodo, ki se nahaja na njej, nato pa gre koda v preverjanje. Da bi si olajšali delo, smo uporabili pretvornik med serijsko in mrežno komunikacijo. S tem smo se izognili implementaciji sklada TCP/IP, leta je namreč implementiran že na samem modulu. Če koda kartice obstaja v bazi uporabnikov, se pošlje pozitiven odgovor. Spet uporabimo prej omenjeni pretvornik, tokrat v obratni smeri. Po tem opravilu vlogo prevzame zaslon na dotik. V primeru, da uporabnik obstaja, mora po približanju kartice za uspešen vstop preko zaslona vnesti še kodo PIN. PIN koda se za vsako kartico določi ročno,

[1] STMicroelectronics, ena največjih svetovnih znamk s polprevodniškimi elementi.

ob vnosu novega uporabnika v bazo. Na enak način kot za kartico se ponovi poizvedba za pripadajoč PIN. Če sta kartica in PIN pravilna, se pošlje signal, ki odklene vrata, v nasprotnem primeru se na ekranu izpiše obvestilo, da je dostop zavrnjen.

Na mikrokrmilniku teče operacijski sistem FreeRTOS. To je operacijski sistem, ki se izvaja v realnem času in je namenjen predvsem vgrajenim sistemom. Posamezne dele programa implementiramo kot opravila.

Poglavje 2

Operacijski sistem

Operacijski sistem je program, ki podpira splošne funkcije računalnika in nudi storitve drugim programom, ki jim pravimo aplikacije. Izvajajo se na računalniku in so zadolžene za zagotavljanje funkcionalnosti, ki jo uporabnik potrebuje oziroma želi. Operacijski sistem je nekakšen posrednik med strojno in programsko opremo. Storitve, ki jih nudi, omogočajo razvijalcem hitrejše pisanje aplikacij, te so preprostejše, zato jih je tudi lažje vzdrževati.

2.1 RTOS

Pri večini operacijskih sistemov se zdi, da se izvaja več programov hkrati. Temu pravimo večopravilnost. Dejansko pa se na posameznem jedru procesorja lahko v trenutku izvaja samo eno opravilo. Delu operacijskega sistema, ki je odgovoren za iluzorno izvajanje več programov hkrati, pravimo razvrščevalnik. Njegova naloga je, da določi program, ki se bo v danem trenutku izvajal, iluzijo hkratnega izvajanja pa doseže s hitrim preklapljanjem (med programi).

Tip operacijskega sistema je definiran glede na način, kako razvrščevalnik določi, kdaj se bo določen program izvajal. Na primer: razvrščevalnik v sistemu, kot je Unix[1], bo zagotovil, da vsak uporabnik dobi približno enako količino procesorskega časa. Drugi primer je operacijski sistem Windows za namizne računalnike, kjer razvrščevalnik skuša uporabniku zagotoviti odzivnost sistema.

[1] Unix = večopravilni večuporabniški računalniški operacijski sistem.

Sistemi v realnem času so zasnovani tako, da se odzivi na zunanje dogodke izvedejo v okviru določenih časovnih mej, ki se jih ne sme prestopiti. To je še posebej pomembno pri vgrajenih sistemih, kjer se pogosto zahteva takojšnja odzivnost. Zagotovimo jo lahko le v primeru, če poznamo obnašanje razvrščevalnika oziroma znamo predvideti njegovo delovanje. Tradicionalni razvrščevalniki operacijskih sistemov v realnem času, kot je FreeRTOS, to zahtevo dosežejo tako, da uporabnikom (programerjem) omogočajo, da vsakemu programu ali opravilu določijo prioriteto izvajanja. Razvrščevalnik potem glede na prioriteto izbere program, ki se bo izvajal.

2.2 Kaj je FreeRTOS

FreeRTOS je različica operacijskega sistem v realnem času, ki je zasnovan tako, da je dovolj majhen za izvajanje na mikrokrmilniku. Potrebno je omeniti, da njegova majhnost ni omejitev, lahko se izvaja tudi drugje. Koliko delovnega pomnilnika potrebuje, je odvisno od aplikacije. Spodaj so prikazani konkretni podatki za:

- IAR STR71x ARM7 port,
- pri polni optimizaciji,
- minimalni konfiguraciji,
- s štirimi različnimi prioritetami opravil.

Del sistema	**Velikost v bajtih**
Razvrščevalnik	236 B
Vsaka vrsta (queue)	vsaj 76 B + prostor za hranjenje vrste
Vsako opravilo	vsaj 64 B + velikost sklada opravila

Potrebna velikost bralnega pomnilnika je odvisna od prevajalnika, arhitekture procesorja in konfiguracije jedra. V zgornjem primeru znaša od 5 do 10 bajtov [2].

Mikrokrmilnik je majhen in z viri omejen procesor, ki na enem čipu vsebuje bralni pomnilnik (ROM ali Flash), kjer se nahaja program, ki se izvaja, in delovni pomnilnik (RAM), ki je potreben za izvajanje programa.

Mikrokrmilniki se večinoma uporabljajo v vgrajenih sistemih, kjer ne vidimo procesorja ali aplikacije same, in imajo ponavadi zelo specifično nalogo. Zaradi omejitev glede velikosti in specifičnosti opravil je implementacija celotnega operacijskega sistema redka oziroma redko sploh mogoča. Zato v osnovi FreeRTOS vsebuje samo razvrščanje v realnem času, medprocesno komunikacijo ter časovnike in orodja za sinhronizacijo. To pomeni, da FreeRTOS natančneje lahko opišemo kot jedro sistema v realnem času. Opravilnosti, kot so konzolni vmesnik za ukaze, omrežni skladi in drugo, lahko po potrebi dodatno vključimo.

FreeRTOS je primeren za aplikacije v realnem času, ki tečejo na majhnih do srednje velikih mikrokrmilnikih. Ponavadi vsebujejo med 32 KB in 512 KB Flash pomnilnika in 16 KB do 256 KB delovnega pomnilnika (RAM). V našem primeru ima MCU STM32F407 1 MB Flash in 192 KB RAM pomnilnika.

Pri aplikacijah v realnem času poznamo dve vrsti časovnih zahtev:

- **trde** (hard real-time requirements) in
- **mehke** (soft real-time requirements).

Pri trdih časovnih zahtevah je prekoračitev roka nesprejemljiva in se odraža v popolni odpovedi sistema. V primeru avtomobilske nesreče nam zračna blazina ne bi prav nič koristila, če bi se sprožila prepozno.

Pri mehkih časovnih zahtevah pa prekoračitev roka ne pomeni odpovedi sistema, ampak povzroči, da je sistem navidezno neodziven. Na primer: počasno odzivanje na pritisk tipk imamo lahko za neodzivnost sistema, čeprav še vedno deluje.

FreeRTOS je jedro, ki deluje v realnem času, in aplikacijam, ki tečejo na njem, omogoča izvrševanje nalog v okviru trdih časovnih zahtev. Prav tako omogoča, da so aplikacije organizirane kot zbirka neodvisnih opravil. Kot smo že prej omenili, se na mikrokrmilnikih, ki imajo samo eno jedro, lahko izvaja samo eno opravilo. Jedro na podlagi prioritete, ki jo določi programer aplikacije, izbere, katero opravilo

se bo izvajalo naslednje. Opravilom, ki imajo trdo časovno zahtevnost, določimo višjo prioriteto kot tistim z mehko. Tako zagotovimo, da se opravila s trdo časovno zahtevnostjo vedno izvedejo pred tistimi z mehko. Seveda pa v večini primerov ne gre tako zlahka. Določanje prioritete opravilom je lahko zapleteno.

2.2.1 Lastnosti FreeRTOSa

Uporaba FreeRTOSa ima sledeče lastnosti:

- preklopne in kooperativne operacije,
- binarni semaforji,
- števni semaforji,
- vrste,
- rekurzivni semaforji,
- zelo fleksibilno določajne prioritet opravilom,
- vzajemno izključevanje (mutex),
- tick sistemski klici,
- idle sistemski klici,
- preverjanje prekoračitve sklada,
- trace sistemski makroji,
- opcijska komercialna licenca in podpora,
- gnezdenje prekinitev (ne za vse arhitekture),
- programsko upravljan sklad za prekinitve (če je to potrebno - varčevanje pri delovnem pomnilniku).

2.2.2 Opravilo (Task)

Opravila so implementirana kot funkcije, ki kot parameter vsebujejo kazalec, vračajo pa nič (void). Prototip funkcije:

void TaskFunction(void *pvParameters);

Vsako opravilo je majhen neodvisen program, z vstopno točko. Ponavadi se izvaja v neskončni zanki in se ne zaključi. Še enkrat poudarimo: opravila v sistemu FreeRTOS se nikoli ne smejo zaključiti ali se izvajati za koncem funkcije, ki opravilo implementira. Če opravilo ni več potrebno/zahtevano, ga enostavno izbrišemo.

Znotraj enega opravila je lahko poljubno število drugih opravil, vsako se izvaja v ločeni niti, s svojim skladom in kopijo vseh avtomatskih spremenljivk znotraj samega opravila.

```
void PrimerOpravila(void *pvParameters)
{
    /* Vsaka instanca opravila ima svojo kopijo spremenljivke i.
    Če bi bila spremenljivka deklarirana kot statična to ne bi bilo res.*/
    int i=0;

    /* Opravilo je ponavadi implementirano v neskončni zanki */
    for(;;)
    {
        /* Koda, ki vsebuje funkcionalnost opravila. */
    }

    /* Če bi opravilo kdaj skočilo iz neskončne zanke ga moramo
    zbrisati preden pride do konca funkcije v kateri je implementirano. */
    vTaskDelete(NULL);
}
```

Slika 2.1: Struktura tipične implementacije opravila

2.3 Jedro sistema v realnem času

Poznamo veliko že ustaljenih tehnik pisanja programske opreme za vgrajene sisteme, kjer se jedra ne uporablja. V preprostejših sistemih je to mogoče najprimernejša rešitev. V kompleksnejših sistemih pa je uporaba jedra bolj smiselna, čeprav odločitev, kdaj uporabiti jedro oziroma ga ne, ostaja v presoji posameznika.

Določanje prioritet posameznim opravilom pomaga zagotoviti izvajanje le-teh znotraj časovnih omejitev, vendar uporaba jedra prinaša tudi druge, manj opazne prednosti.

- Prevzemanje dodatnega dela v povezavi s časom

Jedro je odgovorno za izvrševanje časovnih operacij, hkrati pa ponuja aplikacijski programski vmesnik (API), ki je vezan na časovne operacije. Rezultat je manjša velikost in preprostejša struktura kode celotne aplikacije.

- Vzdrževanje/Razširljivost

S tem ko se izognemo dodatnemu delu v povezavi s časom, se zmanjša odvisnost med posameznimi moduli. Manjša odvisnost omogoča razvoj in morebitno razširjanje aplikacije v bolj predvidljivi in kontrolirani smeri. Ker je jedro odgovorno za časovne operacije, je v primeru, da spremenimo strojno opremo, na kateri se izvaja aplikacija, le-ta manj občutljiva na spremembe.

- Modularnost

Opravila so neodvisni moduli, vsak od njih pa ima natančno definirano nalogo oziroma svoj namen.

- Skupinski razvoj

Opravila morajo imeti natančno definirane vmesnike, da jih lahko razvijamo v skupini.

- Lažje testiranje

Če so opravila natančno definirana, predstavljena kot neodvisni moduli z jasnimi vmesniki, jih lahko testiramo posamično.

- Ponovna uporaba kode

Večja modularnost in manj soodvisnosti rezultira v kodi, ki jo z manjšimi spremembami lahko ponovno uporabimo.

- Izboljšana učinkovitost

Uporaba jedra aplikaciji omogoča dogodkovno vodeno izvajanje. Procesorski čas se ne porablja za preverjanje, če so se določeni dogodki zgodili - ni poolinga. Potrebna koda se izvede samo takrat, ko je dejansko treba nekaj narediti.

Učinkovitost pa se malce zmanjša na račun procesiranja ob vsaki tick[2] prekinitvi in zamenjavi opravila.

- Prosti čas (idle time)

Ko se jedro zažene, se ustvari opravilo *idle* (stanje mirovanja). Izvaja se takrat, ko opravila aplikacije mirujejo. Ima najnižjo prioriteto. Znotraj njega lahko merimo čas, ki ni namenjen procesiranju, lahko izvajamo preverjanja v ozadju ali pa postavimo procesor v stanje najmanjše porabe energije.

- Fleksibilno obravnavanje prekinitev

Prekinitveni strežni programi so lahko zelo kratki. Preko njih se lahko pokliče opravilo, ki za določeno prekinitev opravi večino dela.

- Lažja kontrola nad zunanjimi enotami

Opravila, ki imajo vlogo čuvaja (gatekeeper), so lahko uporabljena za serializacijo dostopa do zunanjih naprav.

[2]tick = sistemski časovnik, to je prekinitev, ki se periodično ponavlja glede na nastavljeno frekvenco. Ob vsaki taki prekinitvi se preveri, če je potrebno preklopiti opravilo.

- Mešane zahteve procesiranja

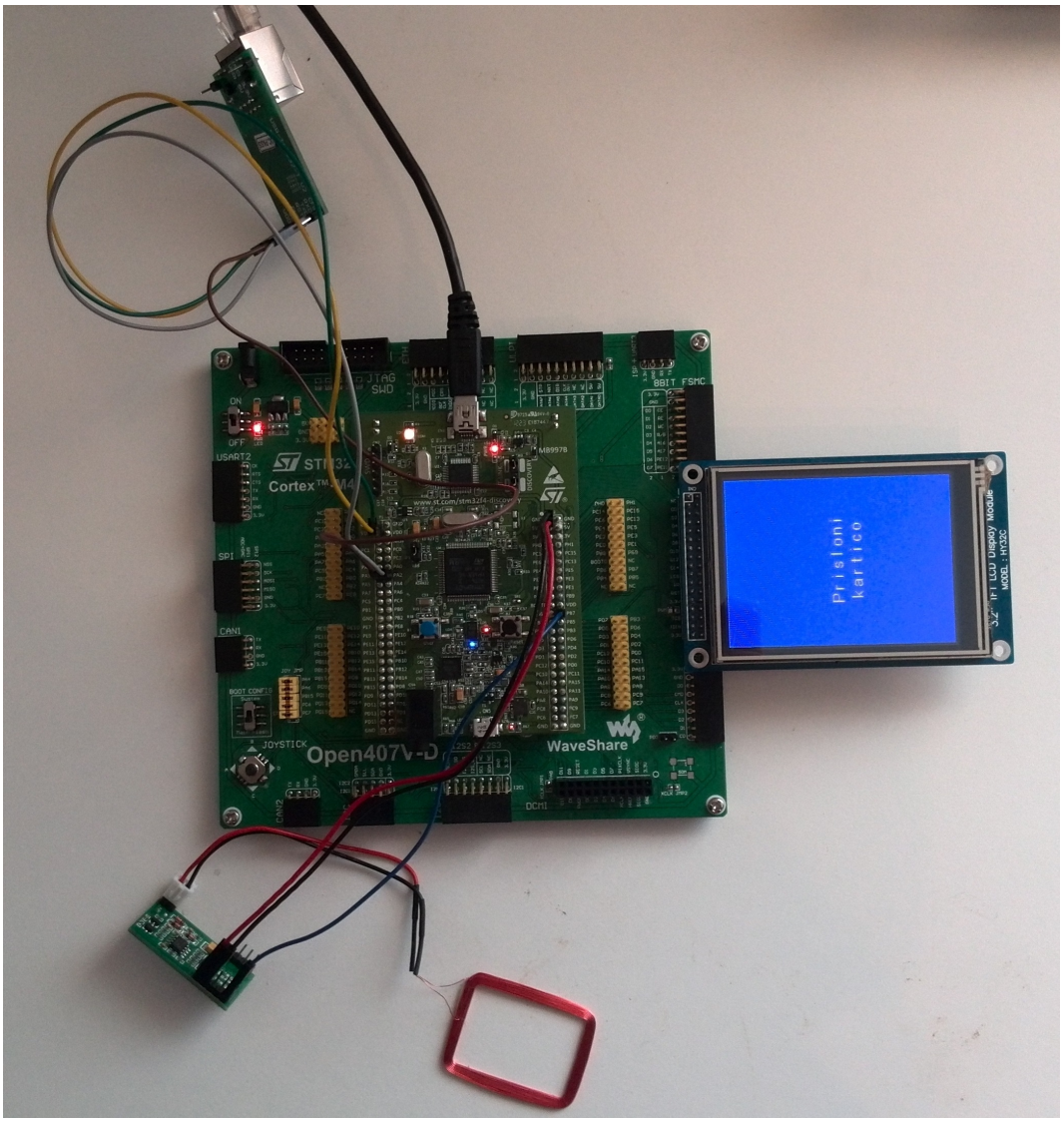
Preproste aplikacije so lahko zasnovane tako, da se v njih pojavljajo zahteve po periodičnem, dogodkovno vodenem ali vztrajnem procesiranju. Pri tem nam pravilna izbira prioritet posameznih opravil in prekinitev omogoča, da zadostimo mehkim in trdim časovnim zahtevam [1].

Poglavje 3

Strojna oprema

Seznam komponent, ki smo jih uporabili pri izdelavi sistema:

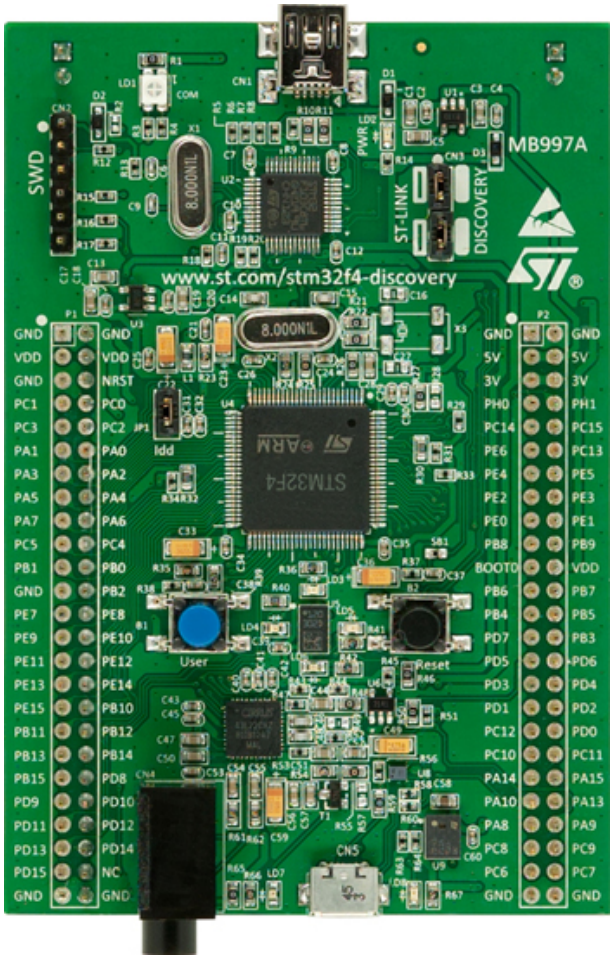
- razvojna ploščica STM32F4Discovery Kit,
- mikrokontroler STM32F407VGT6 (ARM CORTEX-M4),
- barvni 3.2”LCD zaslon na dotik (320x240, 65K barv),
- 125KHz RFID modul,
- pretvornik med serijsko in mrežno komunikacijo USR-TCP232-T V2,
- tiskano vezje (razvojna plošča) Open407V-D,
- RFID pasivne značke.



Slika 3.1: Slika sistema z vsemi komponentami

3.1 STM32F4Discovery Kit

Razvojna ploščica, ki temelji na mikroprocesorju STM32F407VGT6, vsebuje vgrajeno enoto za razhroščevanje ST-LINK/V2, dva MEMS[1] senzorja, digitalni mikrofon in merilnik pospeška, digitalno-analogni avdio pretvornik, svetleče (LED) diode, dva push gumba in USB OTG[2] micro-AB konektor. Napajanje preko USB ali zunanjega vira (3V ali 5V).



Slika 3.2: STM32F4-Discovery Kit. Vir slike [4]

[1]MEMS = mikro elektro-mehanski senzor
[2]USB OTG = USB On-The-Go

3.2 STM32F407

ARM Cortex-M4 je zadnji v vrsti vgrajenih procesorjev ARM, razvitih za preprosto uporabo in učinkovito obdelavo signalov. Porabi malo energije in se uvršča v sam vrh izbire za sisteme, kot so kontrola motorjev, avtomatizacija industrijskih procesov, vgrajeni avdio in drugi sistemi. [6]

Slika 3.3: STM32F407VGT6. Vir slike [5]

Lastnosti:

- ARM Cortex-M4,
- ARM v7E-M arhitektura,
- nabor ukazov:
  - Thumb,
  - Thumb2,
  - DSP razširitve, SIMD nabor ukazov,
  - ukazi za plavajočo vejico (FPv4 razširitev),
- 3-stopenjski cevovod,
- vgrajen prekinitveni krmilnik - do 240 prekinitvenih virov, zakasnitev prekinitev 12 u.p.,

- 210 DMIPS[3] pri 168MHz,
- enota za operande v plavajoči vejici (FPU),
- 1MB pomnilnika Flash,
- 192Kb pomnilnika RAM,
- krmilnik za statični pomnilnik RAM,
- paralelni vmesnik za LCD,
- 2 12-bitna digitalno-analogna pretvornika,
- 3 12-bitne analogno-digitalne pretvornike,
- DMA krmilnik,
- 12 16-bitnih časovnikov, 2 32-bitna časovnika,
- enota za razhroščevanje (SWD[4] in JTAG,[5])
- 140 vhodno-izhodnih portov, do 84MHz,
- 15 komunikacijskih vmesnikov:
  - 3 vmesnike I^2C
  - 4 vmesnike USART
  - 3 vmesnike SPI
  - 2 vmesnika CAN
  - vmesnik SDIO
- 10/100 Ethernet MAC
- CRC enota

[3]DMIPS = Dhrystone MIPS
[4]SWD = Serial Wire Debug
[5]JTAG = Joint Test Action Group

3.3 3.2”TFT LCD HY32C

Slika 3.4: Zaslon na dotik - HY32C. Vir slike [7]

Nekaj podatkov o zaslonu:

Karakteristika	**Tip**
Krmilnik LCD-ja	SSD1289
Krmilnik zaslona na dotik	XPT2046
Vrsta LCD-ja	TFT
Vmesnik LCD-ja	paralelni 16-bitni
Vmesnik zaslona na dotik	SPI
Število barv	65536
Resolucija	320 x 240
Osvetlitev zaslona	LED
Generator napetosti osvetlitve	RT9293

3.3.1 SSD1289 krmilnik

- krmili 76.800 (240x320) elementov oz. pikslov,
- 262.144 različnih barv, 18-bitni RGB (64x64x64),
- vsebuje 172.800 bajtov velik GDDRAM[6].

Do grafičnega pomnilnika dostopamo preko sistemskega vmesnika na paralelni ali serijski način. Pri paralelnem načinu lahko uporabimo dva signalna protokola:

- serija 6800 (8-/9-/16-/18-bit),
- serija 8080 (8-/9-/16-/18-bit).

Pri serijskem načinu uporabimo vmesnik SPI. Na voljo imamo tudi pomožni 18-bitni video vmesnik (VSYNC, HSYNC, DOTCLK, DEN) za animirane slike. [8]

3.3.2 XPT2046 krmilnik

Dekodirno integrirano vezje za rezistivne zaslone na dotik. Njegova funkcija je zaznavanje in pretvorba lokacije pritiska na ekranu. Lastnosti krmilnika:

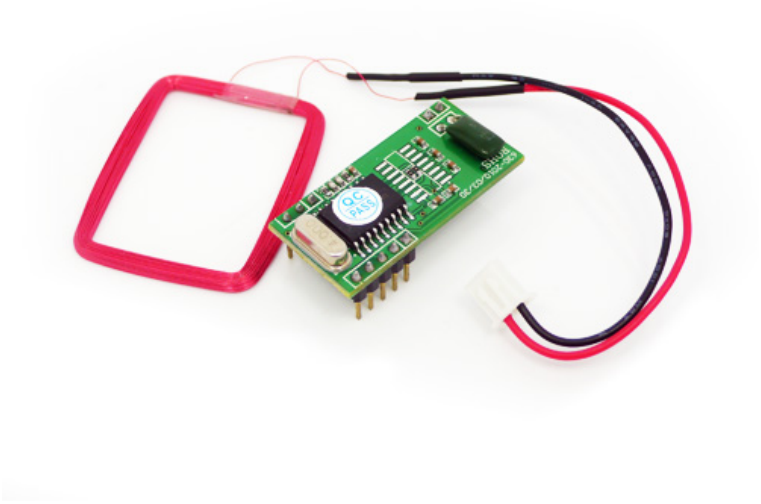
- komunikacija poteka preko vmesnika SPI,
- 12-bitno 125kHz vzorčenje,
- 2 A/D pretvorbi ob pritisku na ekran,
- ob pritisku se proži signal IRQ/PEN, ki je aktiven v nizkem stanju.

Slika 3.5: XPT2046 krmilnik V2. Vir slike [9]

[6]GDDRAM = Graphic Display Data RAM

3.4 125KHz RFID modul

125-kiloherčni RFID modul se uporablja za branje značk, ki delujejo na omenjeni frekvenci. Poleg značk se uporabljajo še kartice, na katere lahko tudi pišemo. Predpogoj za delovanje je, da vsi elementi delujejo na enaki frekvenci.



Slika 3.6: Model RFR101A1M. Vir slike [10]

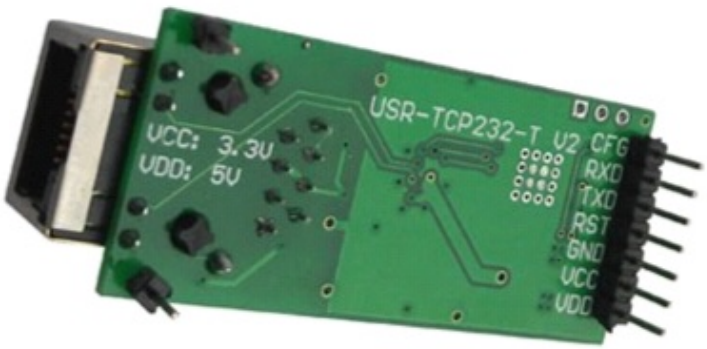
Lastnosti modula:

- možnost priključka zunanje antene,
- maksimalna efektivna razdalja 5 cm,
- čas potreben za dekodiranje je manjši kot 100 ms,
- vmesnik UART TTL,
- podpira EM4100 kompatibilne bralne in bralno-pisalne značke,
- majhen dizajn.

Specifikacije modula:

Frekvenca delovanja	125KHz
Baudna hitrost	9600 (RS232 format)
Vmesnik	Weigang26 ali RS232
Vrsta napajanja	DC 5V (± 5%)
Tok	<50 mA
Območje delovanja	do 50 mm
Velikost	38.5 mm x 19 mm x 9 mm

3.5 USR-TCP232-T



Slika 3.7: USR-TCP232-T V2. Vir slike [11]

USR-TCP232-T je multifunkcijski modul, ki ima implementiran sklad TCP/IP in pošilja podatke iz ethernet vrat na serijska vrata ter obratno. Vgrajene naprave tako s preprosto vključitvijo modula dobijo funkcijo omrežne naprave. Modul se lahko nastavi na 4 različne načine delovanja, in sicer kot odjemalec TCP, strežnik TCP, odjemalec UDP ali strežnik UDP.

Lastnosti modula:

- 10/100 ethernet,
- podpora za AUTO MDI[7]/MDIX[8]; za povezavo se lahko uporabi crossover ali paralelni kabel,
- baudna hitrost od 300 do 25600 bps,
- delovni način: strežnik ali odjemalec TCP, strežnik ali odjemalec UDP,
- nastavljanje delovnega načina preko omrežja ali serijskega porta,
- lahko deluje kot virtualen COM port,
- avtomatska vzpostavitev povezave TCP.

Opis pinov:

Pin	Funkcija	Opis
VDD	Napajanje 1	4.5 $\sim$ 5.5 V
VCC	Napajanje 2	3.3 V
GND	Ozemljitev	Ozemljitev
RST	Reset	Reset modula - 200 ms v nizkem stanju
TXD	Pošiljanje	RS232 prenosni port, TTL 3.3 V in 5 V
RXD	Prejemanje	RS232 prejemni port, TTL 3.3 V in 5 V
CFG	Konfiguracija	Konfiguracija v nizkem stanju

[7]MDI = Medium Dependent Interface
[8]MDIX = Medium Dependent Interface Crossover

3.6 Razvojna plošča Open407V-D

Open407V-D je razvojna plošča za 32-bitne mikrokrmilnike STM, ki temeljijo na 32-bitnem procesorju ARM. Razvita je posebej za STM32F4DISCOVERY Kit, ki vsebuje mikrokrmilnik STM32F407VGT6.

Slika 3.8: WaveShare Open407V-D. Vir slike [12]

Oblika plošče temelji na odprtosti in modularnosti, kar omogoča priklop različnih razširitvenih modulov. Izkaže se kot zelo primerna za začetek razvoja aplikacij za mikrokrmilnike STM32F4.

Seznam komponent, ki jih plošča vsebuje:

- STM32F4DISCOVERY vtič,
- vmesnik USART2, na katerega se lahko priklopi RS232, RS485, USB TO 232, itd.,
- vmesnik SPI1/SPI2 + AD/DA,
  - preprosta povezava do SPI zunanjih enot, kot so SD kartice, MP3 moduli, LCD zasloni, itd.,
  - SPI1 omogoča AD/DA alternativno funkcijo, priklop AD/DA konverter modula,
- vmesnik CAN1, CAN2,
- vmesnik I^2C1/I^2C2,
- $I^2S2/I^2S3/I^2C1$ za povezavo do I^2S zunanjih naprav,
- vmesnik DCMI za priklop kamere,
- vmesnik SDIO za priklop Micro SD modula, veliko hitrejši dostop kot prek SPI,
- vmesnik FSMC + SPI (16-bitni FSMC) za priklop LCD zaslona na dotik,
- vmesnik FSMC (8-bitni FSMC) za priklop omrežnih naprav, NandFLash, itd.,
- vmesnik USART3,
- vmesnik ULPI za priklop visoko hitrostnih zunanjih USB naprav,
- vmesnik Ethernet za priklop mrežnega modula,
- 5V DC vtič,
- 5V/3.3V vhod/izhod,
- MCU pini,
- vmenik JTAG/SWD za razhroščevanje.

3.7 RFID pasivne značke

Tehnologija RFID oziroma Radio Frequency IDentification obstaja že več kot 50 let. Identifikacija se izvaja preko elektromagnetnega valovanja na področju radijskih frekvenc. RFID značke lahko prepoznamo brezkontaktno, kar je ena glavnih prednosti omenjene tehnologije.

Slika 3.9: Pasivna značka (tag). Vir slike [14]

Ločimo pasivne in aktivne značke. Sestavljene so iz majhnega integriranega vezja in antene. Pri pasivnih značkah se energija za napajanje pridobi iz radijskih valov čitalca. Pogoj, da komunikacija steče je, da sta čitalec in značka dovolj blizu. Aktivne značke imajo poleg vezja in antene še svoj lasten vir napajanja - baterijo, zato jim za oddajanje podatkov ni treba čakati, da pridejo v doseg čitalca. Večina značk, ki je v uporabi danes je pasivnih. Frekvence na katerih deluje tehnologija RFID so različne, od 120 kHz pa vse do 10 GHz. Za komunikacijo je pomembno, da vse komponente delujejo na enaki frekvenci. Čitalec, ki deluje na frekvenci 13,56 MHz ne zazna značke, ki deluje na frekvenci 125 kHz in obratno. Frekvenca se izbere glede na namen uporabe. Pri višjih frekvencah sta domet in hitrost prenosa podatkov večja, prav tako pa so dražje tudi značke. Pojavljajo se v obliki etiket, kartic, obeskov, škatlic itd. Uporabljajo se za sledenje živali, prepoznavanje artiklov, ladijskih kontejnerjev, pri izposoji knjig, za kontrolo dostopa...

Poglavje 4

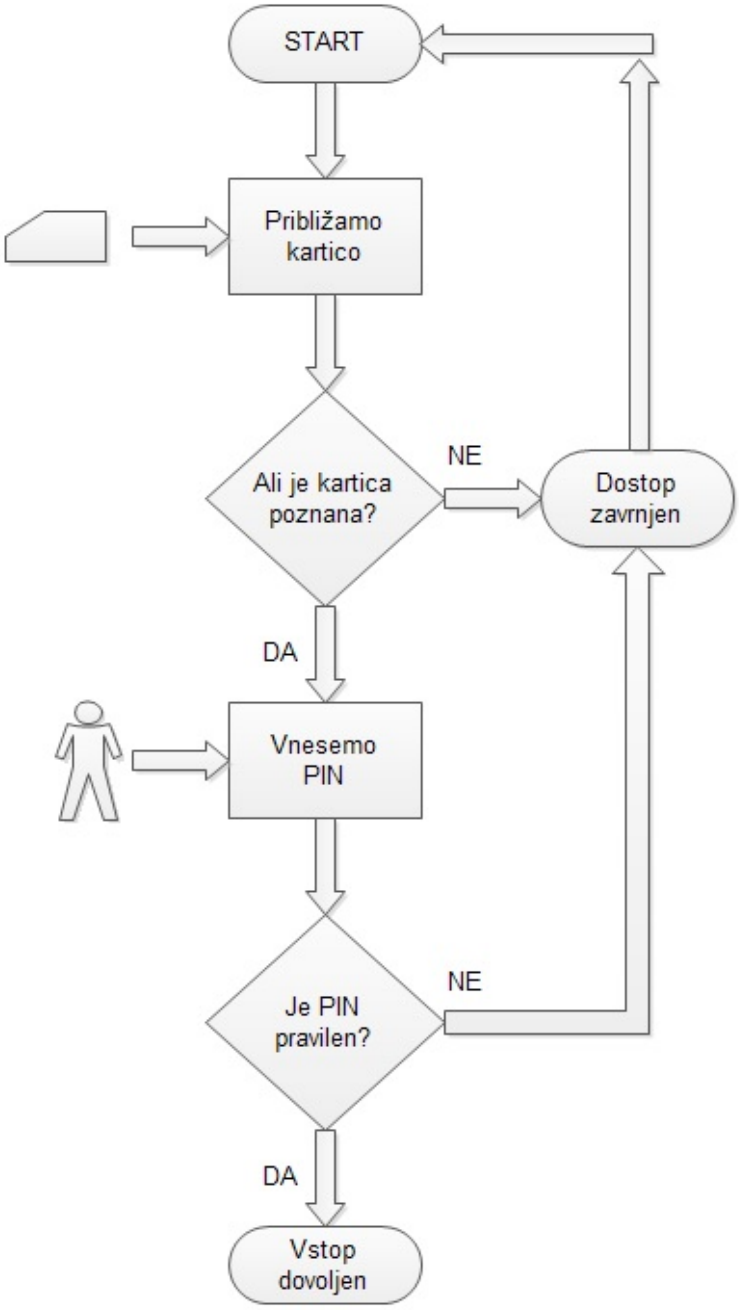
Programski del

Programski del celotnega projekta se deli na dva dela:

- koda na mikrokrmilniku (C) ter
- strežniški program in program za urejanje uporabnikov (Java in C#).

Najprej je opis programa, ki se izvaja na mikrokrmilniku. Bistvo tvori šest opravil, ki jih bomo podrobneje predstavili, vsako ima svojo prioriteto in se izvaja samo po potrebi. Sledi vmesnik za delo z uporabniki. Preko njega lahko le-te dodajamo ali brišemo, na voljo pa imamo še dnevnik dostopa. Na koncu je predstavitev strežniškega programa. Ta se izvaja kot proces v ozadju in skrbi za dostopanje do baze podatkov, v kateri se beležijo vsi dostopi in izvajajo preverjanja PIN kod ter številk pasivnih kartic oziroma značk.

Diagram poteka celotnega postopka



Slika 4.1: Diagram poteka

4.1 Mikrokrmilnik - opravila

Opravila v sistemu FreeRTOS so dogodkovno vodena. To pomeni, da se izvajajo (procesirajo) samo takrat, kadar se zgodi dogodek, ki sproži njihovo izvajanje. Pred tem opravilo ne more biti v stanju izvajanja. Stanja, v katerih se opravila lahko nahajajo, so naslednja:

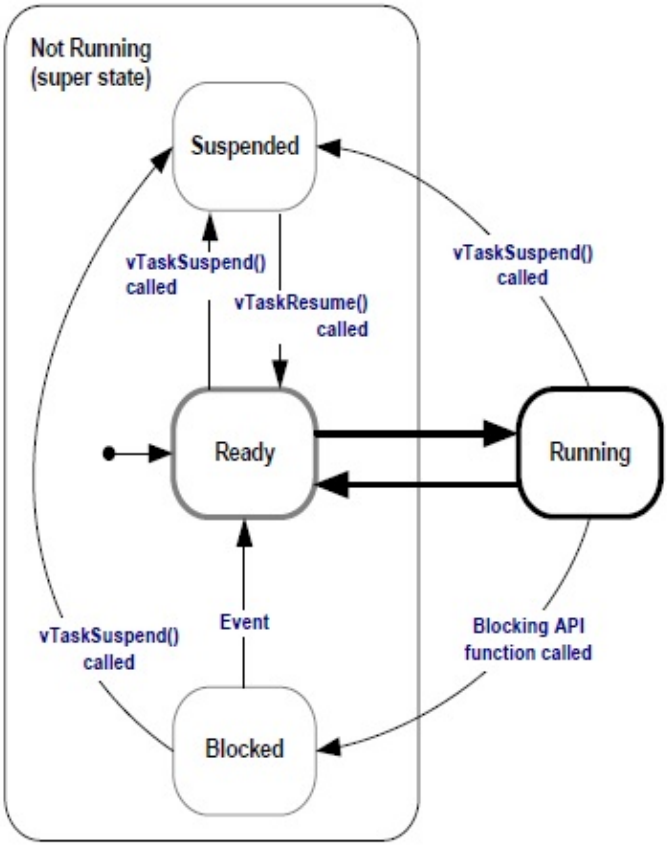
- blokirano (Blocked),
- pripravljeno na izvajanje (Ready),
- začasno odstranjeno (Suspended),
- v izvajanju (Running).

Opravilo, ki čaka na določen dogodek, je v stanju **blokirano**. Čaka lahko na dve vrsti dogodkov:

1. **Sinhronizacijski dogodek** - dogodek se zgodi v drugem opravilu ali prekinitvi. Na primer, kadar čaka, da se v vrsti (queue) pojavijo podatki. Poleg vrst se za namen sinhronizacije lahko uporabi tudi vzajemno izključevanje (mutex) ter binarne, števne in rekurzivne semaforje. V našem programu smo uporabili binarne semaforje.
2. **Časovni dogodek** - pretečeni čas ali dosežen absolutni čas. Preden opravilo zapusti stanje "blokirano", počakamo, da preteče 10 milisekund (zamik).

Naslednje stanje, **začasno odstranjeno**, povzroči, da opravilo ni dostopno razvrščevalniku. Vanj lahko opravilo vstopi samo s klicem vTaskSuspend() funkcije, iz njega pa s klicem vTaskResume(). Večina aplikacij tega stanja ne uporablja, v našem programu pa en tak klic obstaja.

Opravila, ki so v stanju neizvajanja, hkrati pa niso blokirana oz. začasno odstranjena, se nahajajo v stanju **pripravljeno na izvajanje**. Pomeni, da je njihovo izvajanje mogoče in so pripravljena, ampak se trenutno ne izvajajo. [1]



Slika 4.2: Končni avtomat stanj opravil [1]

Seznam opravil v našem programu:

1. accessDenied
2. accessApproved
3. showIdleScreen
4. verifyRFID
5. showKeyboard
6. highlightNumber

4.1.1 accessDenied in accessApproved

Preprosti opravili, ki čakata na ustrezen semafor **xRfidNotValid** oz. **xRfidValid** in prikažeta zaslon z napisom "Vstop zavrnjen" oz. "Vstop odobren". Razlika je v barvi ozadja, v prvem primeru je rdeča, v drugem zelena. Prioriteta opravila je 2.

(a) Odobreno

(b) Zavrnjeno

Slika 4.3: Prikaz zaslonov obeh opravil

```
1  void accessDenied(void *pvParameters)
2  {
3      for(;;) {
4          //opravilo je blokirano, dokler ne dobimo ustreznega
5          //semaforja
6          configASSERT(xRfidNotValid);
7          xSemaphoreTake(xRfidNotValid, portMAX_DELAY);
8
9          repaint = 1;
10         repaintKeyboard = 1;
11         LCD_Clear(RED);
12         LCD_SetTextColor(BLACK);
13         LCD_SetBackColor(RED);
14         LCD_CharSize(24);
15         LCD_StringLine(60, 110, vstop);
16         LCD_StringLine(56, 130, zavrnjen);
17         Delay(25000000);
18     }
19 }
```

Listing 4.1: opravilo accessDenied

4.1.2 showIdleScreen

Opravilo z najnižjo prioriteto. Izvaja se, ko so vsa druga blokirana oz. začasno odstranjena. Na modri podlagi prikazuje potujoč rumen napis "Prisloni kartico". Edino opravilo, ki za izvajanje ne potrebuje semaforja. Prioriteta opravila je 1.

Slika 4.4: Prikaz zaslona opravila showIdleScreen

```
1  void showIdleScreen(void *pvParameters)
2  {
3      int i, k;
4      LCD_Clear(BLUE);
5      LCD_SetTextColor(YELLOW);
6      LCD_SetBackColor(BLUE);
7      LCD_CharSize(24);
8
9      for(;;) {
10         for(i = -40; i < 320; i++) {
11             if(repaint) {
```

```
12                      LCD_Clear(BLUE);
13                      LCD_SetTextColor(YELLOW);
14                      LCD_SetBackColor(BLUE);
15                      LCD_CharSize(24);
16                      repaint = 0;
17                  }
18                  while(k > 0)
19                      k--;
20              LCD_StringLine(60, i, prisloni);
21              LCD_StringLine(66, i+27, kartico);
22              k = 200000;
23          }
24      }
25  }
```

Listing 4.2: opravilo showIdleScreen

4.1.3 verifyRFID

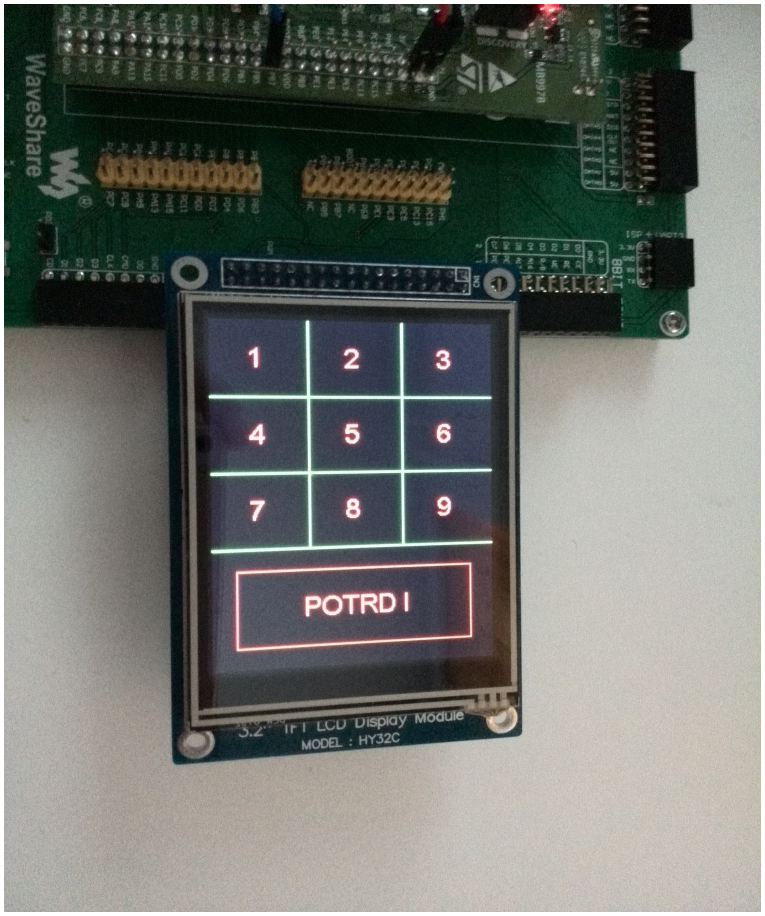
Ko kartico približamo čitalcu, se sproži prekinitev, v kateri se prebere koda kartice. Nato se pošlje zahteva po preverjanju, če koda v bazi obstaja. Dokler ne dobimo semaforja **xUsartRead**, je opravilo blokirano - primer uporabe binarnih semaforjev za namen sinhronizacije. Prioriteta opravila je 2.

```
1  void verifyRFID(void *pvParametrs)
2  {
3      for(;;) {
4          configASSERT(xUsartRead);
5          xSemaphoreTake(xUsartRead, portMAX_DELAY);
6
7          USART_puts(USART2, prebrana);
8      }
9  }
```

Listing 4.3: opravilo verifyRFID

4.1.4 showKeyboard

Opravilo, ki čaka na semafor **xRfidValid** in na ekranu prikaže tipkovnico za vnos PIN-a. Prioriteta opravila je 3. Tukaj se semafor zavzame samo enkrat - na začetku izven neskončne zanke. To pomeni, da se opravilo potem, ko semafor ima, izvaja v nedogled. Neskončno izvajanje preprečimo z uporabo časovnika in funkcije, ki se kliče po preteku le-tega. Periodo smo nastavili na 10 sekund. Če v tem času uporabnik ne vnese PIN-a, se strežniku pošlje neveljaven PIN, opravilo pa začasno odstrani.



Slika 4.5: Prikaz zaslona opravila showKeyboard

```
1  void showKeyboard(void *pvParametrs)
2  {
3      configASSERT(xRfidValid);
4      xSemaphoreTake(xRfidValid, portMAX_DELAY);
5
6      pinEnterAllowed = 1;
7      repaint = 1;
8      LCD_Init();
9      LCD_Clear(BLACK);
10     LCD_SetTextColor(GREEN);
11     LCD_SetBackColor(BLACK);
12     LCD_CharSize(24);
13
14     for(;;) {
15         //pogledamo ce timer ze tece, drugace ga startamo
16         if(!xTimerIsTimerActive(pinEnterTime))
17             xTimerStart(pinEnterTime, 50);
18
19         .
20         .
21         .
22
23         if(PINtapsCount == 4) {
24             pinEnterAllowed = 0;
25             PINtapsCount = 0;
26             repaint = 1;
27             vTaskSuspend(showKeyboardHandle);
28             xTimerStop(pinEnterTime, 100);
29             USART_puts(USART2, PIN);
30
31             //delay zaradi zamika v komunikaciji
32             Delay(2000000);
33         }
34     }
35 }
```

Listing 4.4: opravilo showKeyboard

4.1.5 vTimerCallback

Funkcija, ki se kliče po preteku časa 10 sekund. Ustavimo časovnik, ponastavimo vse relevantne spremenljivke, začasno odstranimo opravilo in pošljemo neveljaven PIN.

```
1  void vTimerCallback(xTimerHandle pxTimer)
2  {
3      xTimerStop(pxTimer, 100);
4
5      //ce v casu 10s nismo vnesli celotnega pina, se serverju
6      //poslje PIN 0000, potrebne spremenljivke resetira
7      if (PINtapsCount != 4) {
8          char falsePIN[] = {'0','0','0','0'};
9          PINtapsCount = 0;
10         pinEnterAllowed = 0;
11         touchPressed = 0;
12         acknowladgePIN = 0;
13         repaintKeyboard = 1;
14         repaint = 1;
15
16         //zacasno odstranimo opravilo, ki prikazuje tipkovnico
17         //in strezniku posljemo neveljaven pin
18         vTaskSuspend(showKeyboardHandle);
19         USART_puts(USART2, falsePIN);
20         Delay(2000000);
21     }
22 }
```

Listing 4.5: callback funkcija časovnika

4.1.6 highlightNumber

Zadnje opravilo opravi nalogo osvetlitve tipke, ki smo jo pritisnili na tipkovnici. Njegova prioriteta je 4. Za začetek izvajanja čaka na semafor **xReadPin**.

```
1  void highlightNumber(void *pvParametrs)
2  {
3      for (;;) {
4          configASSERT(xReadPin);
5          xSemaphoreTake(xReadPin, portMAX_DELAY);
6
7          if (touchPressed) {
8              //pritisk na 1
9              if (PINtapsCount < 4 && (Pen_Point.X0 >= 1 &&
                   Pen_Point.X0 <= 80) && (Pen_Point.Y0 >= 251 &&
                   Pen_Point.Y0 <= 320)) {
10                 PIN[PINtapsCount++] = '1';
11                 areaPressed(1);
12                 LCD_PutChar(30, 25, pinKeyboard[1]);
13                 Delay(1000000L);
14                 areaPressed(0);
15                 LCD_PutChar(30, 25, pinKeyboard[1]);
16                 touchPressed = 0;
17             }
18             .
19             .
20             .
21         }
22     }
23 }
```

Listing 4.6: opravilo highlightNumber

4.2 Mikrokrmilnik - prekinitve

V programu se pojavijo 3 vrste prekinitev:

- ko približamo kartico čitalcu (USART1),
- ko dobimo odgovor strežnika (USART2),
- ko vnesemo PIN, pritisnemo na ekran (SPI1).

Pri branju kartice se začetek označi s start bajtom (zastavico), ki ima vrednost 0x02. Nato sledi 12 ASCII [1] znakov in stop bajt 0x03. Ko se prebrana koda pošlje v preverjanje, strežnik odgovori z 0x31 ali 0x32. Koda je poznana oz. je ni v bazi. USART2 vmesnik smo tako uporabili za dvosmerno komunikacijo, medtem ko USART1 uporabljamo samo za branje. Pritisk na ekran proži prekinitev preko SPI1 vmesnika. Odgovor ali je PIN pravilen ali ne prav tako preberemo preko USART2. Strežnikov odgovor je lahko 0x33 - PIN pravilen ali 0x34 - PIN napačen.

Prekinitveni strežni programi za omenjene prekinitve:

- void USART1_IRQHandler(void),
- void USART2_IRQHandler(void),
- void EXTI9_5_IRQHandler(void).

[1] ASCII = American Standard Code for Information Interchange, 7-bitni nabor znakov

4.2.1 USART1 handler

```
1  void USART1_IRQHandler(void)
2  {
3      unsigned char sign = '\0';
4      sign = USART_ReceiveData(USART1);
5
6      portBASE_TYPE xHigherPriorityTaskWoken;
7      xHigherPriorityTaskWoken = pdFALSE;
8
9      if(sign == 0x03) {
10         USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
11         ETX = 1;
12         k = 0;
13         xSemaphoreGiveFromISR(xUsartRead,
14         &xHigherPriorityTaskWoken);
15         portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
16     }
17
18     if(!ETX)
19         prebrana[k++] = sign;
20
21     if(sign == 0x02) {
22         ETX = 0;
23         k = 0;
24     }
25 }
```

Listing 4.7: PSP za USART1

4.2.2 USART2 handler

```
1  void USART2_IRQHandler(void)
2  {
3      unsigned char sign = '\0';
4      sign = USART_ReceiveData(USART2);
5
6      portBASE_TYPE xHigherPriorityTaskWoken;
7      xHigherPriorityTaskWoken = pdFALSE;
8
9      USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
10
11     if(sign == 0x31) {
12         repaintKeyboard = 1;
13         pinEnterAllowed = 1;
14         //prestavimo displayKeyboard task v READY stanje
15         vTaskResume(prikaziTipkovnicoHandle);
16
17         xSemaphoreGiveFromISR(xRfidValid,
18         &xHigherPriorityTaskWoken);
19         portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
20     }
21     else if(sign == 0x32 || sign == 0x34){
22         xSemaphoreGiveFromISR(xRfidNotValid,
23         &xHigherPriorityTaskWoken);
24         portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
25     }
26     else {
27         xSemaphoreGiveFromISR(xPinValid,
28         &xHigherPriorityTaskWoken);
29         portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
30     }
31 }
```

Listing 4.8: PSP za USART2

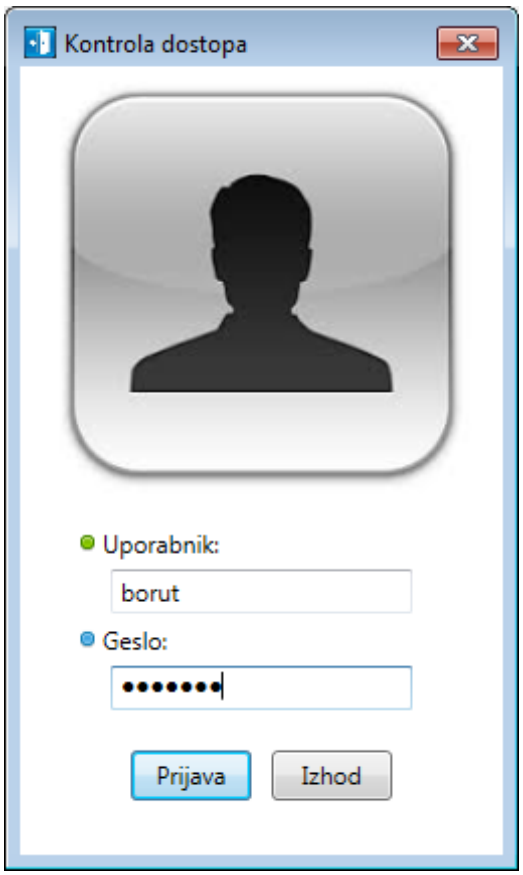
4.2.3 EXTI9_5 handler

```
1  void EXTI9_5_IRQHandler(void)
2  {
3      //ce ne dovolimo vnosa pina, ignoriramo vsak pritisk na
           ekran
4      if(!pinEnterAllowed)
5          EXTI_ClearITPendingBit(EXTI_Line5);
6
7      else {
8          portBASE_TYPE xHigherPriorityTaskWoken;
9          xHigherPriorityTaskWoken = pdFALSE;
10
11         touchPressed = 1;
12         Convert_Pos();
13
14         //zamik, da ne zazna vec dotikov "hkrati"
15         Delay(3000000L);
16         EXTI_ClearITPendingBit(EXTI_Line5);
17
18         xSemaphoreGiveFromISR(xReadPin,
19         &xHigherPriorityTaskWoken);
20         portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
21     }
22 }
```

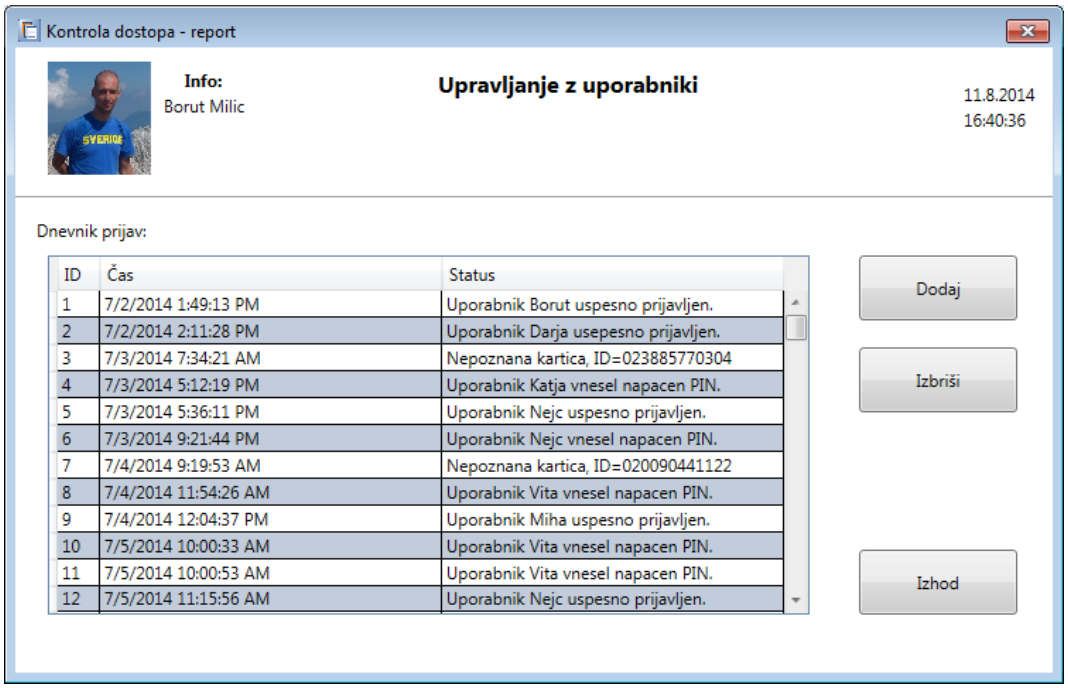
Listing 4.9: PSP za SPI1

4.3 Upravljanje z uporabniki

Za potrebe upravljanja z uporabniki smo napisali dodaten program v programskem jeziku C#. Ima preprost grafični vmesnik, dve modalni okni ter tabelo (dnevnik), kjer se beležijo vsi uspešni in neuspešni dostopi.

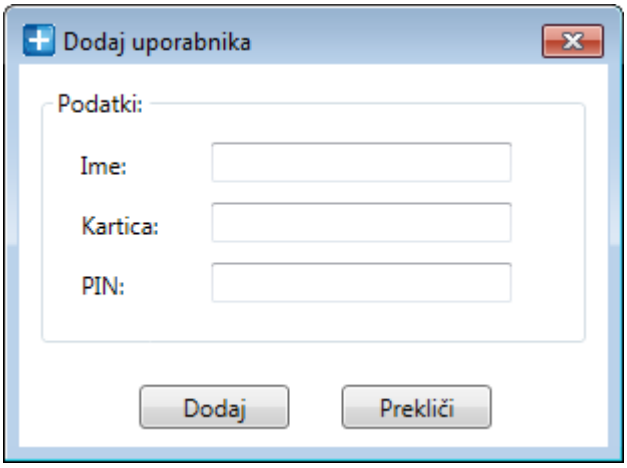


Slika 4.6: Prijavno okno

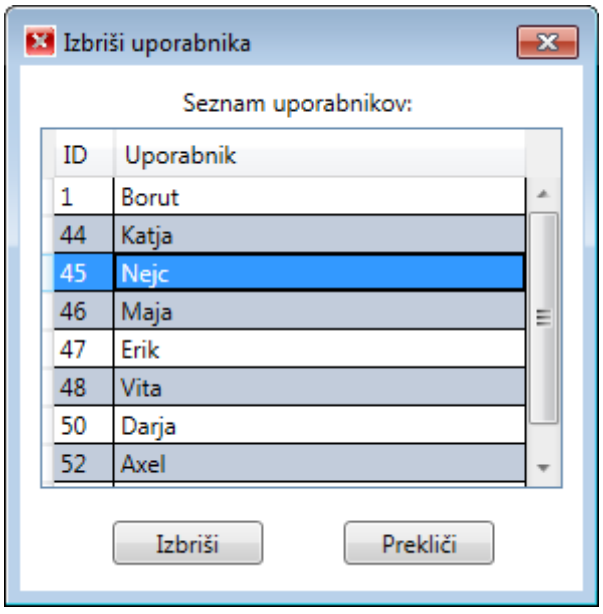


Slika 4.7: Glavno okno, upravljanje z uporabniki

V ozadju se izvaja še strežniški program, na katerega se povezuje mikrokrmilnik preko modula USR-TCP232. Uporabljena podatkovna baza je MySQL in je sestavljena iz dveh tabel. V prvi imamo podatke o uporabnikih, v drugi pa so zapisi o dogodkih, ki se prikazujejo v dnevniku. Program je napisan v programskem jeziku Java.



Slika 4.8: Dialog za dodajanje



Slika 4.9: Dialog za brisanje

Poglavje 5

Sklepne ugotovitve

Rezultat diplomske naloge je sistem, ki deluje, vendar konkretna implementacija na vratih še ni bila izvedena. Ker gre za prvi tovrstni projekt, je narejen dokaj preprosto. Pri komunikaciji med strežnikom in čitalcem kartic ni enkripcije. Nekaj težav smo imeli pri pošiljanju kode kartice iz mikrokrmilnika na strežnik, rešili pa smo jo tako, da smo pošiljali znak za znakom in ne celoten niz. Rešitev je sprejemljiva, ker je koda dolga samo 12 znakov. Pomembna je vloga časovnika, 10 sekund, ki jih ima uporabnik na voljo za vnos PIN-a. Ena možnost je bila, da se na strani strežnika uporabi Java NIO[1] API, druga, to smo tudi implementirali, pa je programski števec mikrokrmilnika, ki po preteku desetih sekund strežniku sam pošlje neveljaven PIN.

Operacijski sistem FreeRTOS za tako majhen projekt morda niti ni potreben, smo pa z njegovo vključitvijo spoznali nov vidik programiranja naprav, ki nam bo prišel prav v prihodnje, ko se bodo razvijali bolj kompleksni sistemi. Sam koncept opravil je zelo jasen, nekoliko pazljivosti in premisleka pa je potrebno pri določanju prioritet. ARMova Cortex-M jedra imajo, za razliko od večine drugih, inverzno relacijo med številčnimi in logičnimi vrednostmi prioritet. To pomeni, da ima prekinitev ali opravilo s prioriteto 1 prednost pred tistimi, ki imajo prioriteto 2 ali več.

Sedaj, ko je prvi projekt za nami, že razmišljamo o nadaljnih, saj so strojna oprema oz. razširitveni moduli, ki se priključijo na razvojno ploščo, lahko dostopni

[1] NIO, Non-blocking Input/Output

in ne pretirano dragi. Je pa malce škoda, da smo področje vgrajenih naprav spoznali šele na koncu študija, v zadnjem semestru. Vsekakor nas programiranje na tem področju veseli veliko bolj, kot pa razvijanje "splošne" programske opreme, kjer se produkta ne da "otipati".

Literatura

[1] R. Barry. *Using the FreeRTOS^{TM}Real Time Kernel, A Practical Guide*, dostopno na:
http://shop.freertos.org/RTOS_primer_books_and_manual_s/1819.htm

[2] FreeRTOS FAQ - Memory Usage, Boot Times & Context Switch Times, dostopno na:
http://www.freertos.org/FAQMem.html#RAMUse

[3] STM32F4DISCOVERY, dostopno na:
http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419

[4] Slika STM32F4DISCOVERY Kit, dostopno na:
http://www.st.com/st-web-ui/static/active/en/fragment/product_related/rpn_information/board_photo/stm32f4_discovery.jpg

[5] Slika STM32F407VGT6, dostopno na:
http://i00.i.aliimg.com/wsphoto/v0/629453324/STM32F407VGT6-new-and-original-ROMON-reflective-photoelectric-and-infrared-sensor.jpg_250x250.jpg

[6] ARM Cortex-M4, dostopno na:
http://www.arm.com/products/processors/cortex-m/cortex-m4-processor.php

[7] 3.2”320x240 Touch LCD C, dostopno na:
http://www.wvshare.com/product/3.2inch-320x240-Touch-LCD-C.htm

[8] Krmilnik SSD1289, dostopo na:
http://www.solomon-systech.com/en/product/display-ic/smart-tft-lcd-driver-controller/ssd1289/

[9] Slika krmilnika XPT2046, dostopno na:
http://ecx.images-amazon.com/images/I/41MwS8HudjL.jpg

[10] Slika 125KHz RFID modula, dostopno na:
http://www.seeedstudio.com/depot/images/product/P1240147.jpg

[11] Slika pretvornega modula USR-TCP232-T, dostopno na:
http://www.tcp232.net/image/cache/data/New/TCP%20series/embedded%20uart%20module-usr-tcp232-T-500x500.jpg

[12] Slika Open407V-D, dostopno na:
http://www.wvshare.com/img/preview/Open407V-D-Standard_l.jpg

[13] STM32F407VG, dostopno na:
http://www.st.com/web/catalog/mmc/FM141/SC1169/SS1577/LN11/PF252140

[14] Slika RFID značke, dostopno na:
http://www.avc-shop.de/WebRoot/Store15/Shops/64272905/5312/90CE/7DF0/ADC5/20E8/C0A8/2BBA/AF16/RFID1.JPG

[15] WaveShare Open407V-D, dostopno na:
http://www.wvshare.com/product/Open407V-D-Standard.htm

[16] Tehnologija RFID, dostopno na:
http://www.ukm.uni-mb.si/UserFiles/658/File/RFID.pdf